S/H circuit multiplexes op amp

Tarlton Fleming
Maxim Integrated Products, Sunnyvale, CA

The sample-and-hold (S/H) circuit in **Fig 1** costs only $3.50 (1000) because it switches its single op amp between two functions. The op amp buffers the input (V_{IN}) while the circuit is in sample mode and buffers the hold capacitor, C_H, while the circuit is in hold mode.

The two digital inputs are compatible with TTL and CMOS logic levels. Input $\overline{S}$/H controls the circuit's operating mode (low is sample), and $\overline{DISCH}$ is an optional control input whose low state commands a rapid and complete discharge of C_H.

You can use a general-purpose op amp for IC_1, provided its input bias current is acceptable. Bias current usually dominates the hold-mode droop rate. C_H can range from 100 pF to 0.1 μF. When driving such a capacitive load, most op amps will oscillate without an isolating resistor, such as R_1, of 100 to 200Ω in their feedback loops.

Typical performance with a 0.01-μF hold capacitor includes a droop rate of ≤100 mV/sec, aperture time of ≤100 nsec, an offset voltage of ≤5 mV, output charge injection of ≤5 pC, and an acquisition time of ≤1 μsec (for ±10% accuracy) or ≤5 μsec (±0.1% accuracy).

Performance is about the same for ±15 or ±12V supplies, and the system also works well on a unipolar supply of 10 to 30V. Whatever the supply's configuration, the op amp's common-mode range restricts V_{IN} to about 2V less than the supply rails. The control inputs' switching thresholds remain the same regardless of supply levels.

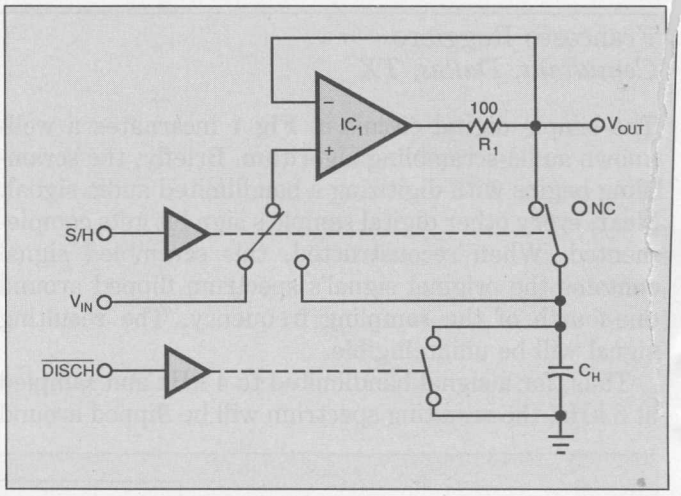

Fig 1—Elements of a quad analog switch *hardware-multiplex an op amp between input and hold-capacitor buffering functions in this economical S/H circuit.*

(EDN BBS DI #888)

To Vote For This Design, Circle No. 747

8051 routine divides quickly

Ron Mowrer
Rocky Mountain Instrument, Thermopolis, WY

The program in **Listing 1** speeds some division operations by taking advantage of the 8051 single-chip μP's byte-divide instruction. This program embodies a nibble-mode algorithm and handles operations such as division by a constant. The routine in the **listing** will work with any number of bytes in the dividend but restricts the divisor's range to 4 bits or less.

First, the nibble divisor divides into the dividend's most-significant byte. Next, the remainder from that division (which is never greater than a nibble) combines with the next-most-significant *nibble* of the dividend for the next division. The algorithm repeats until the dividend is completely "nibbled" up.

The routine in the **listing** handles 3-byte dividends. To accommodate different-length dividends, simply change the loop counter's initial value from 2, in line 165, to one less than the total number of bytes in your

dividend. You can alter the routine to either load the MathPtr (line 157) and byte count directly, or have the calling routine pass these parameters to this routine.

Although ideal for such tasks as dividing by 10, the routine's speed, compared to normal division algorithms, makes it attractive for dividing by larger divisors that break down into nibble divisors. For example, calling the routine with 5 and then 7 will equal a division by 35 (5×7) and three calls with a divisor of 5 is the same as dividing by 125 (5×5×5).

(EDN BBS DI #890) EDN

To Vote For This Design, Circle No. 748

Listing 1—Fast nibble-mode division routine

```
                    001     $ INCLUDE (REG451.PDF)
                =1  002 +1  $ NOLIST
                    136
                    137     PUBLIC  DivNib
                    138     EXTRN   DATA (MathPtr)
                    139
  0010              140     MathBank        EQU     00010000B       ;PSW VALUE FOR REGISTER BANK 2
                    141
                    142     USING 2
                    143
                    144     ;-----------------------------------------------------------------
                    145     ;  DIVIDE 3 BYTE BINARY NUMBER BY NIBBLE
                    146     ;       INPUT:  MathPtr = PTR TO MS BYTE OF 3 BYTE DIVIDEND/RESULT
                    147     ;               ACC = NIBBLE DIVISIOR ( 1H <= DIVISOR <= OFH )
                    148     ;       OUTPUT: RESULT IN SAME LOCATION AT MathPtr
                    149     ;               ACC = REMAINDER
                    150     ;-----------------------------------------------------------------
                    151
                    152     DivNib:
  0000 C0F0         153             PUSH    B               ;SAVE B & PSW REGISTERS ON STACK
  0002 C0D0         154             PUSH    PSW
  0004 75D010       155             MOV     PSW,#MathBank   ;SWITCH TO REGISTER BANK 2
  0007 FB           156             MOV     R3,A            ;STORE NIBBLE DIVISOR
  0008 A800     F   157             MOV     R0,MathPtr      ;R0 NOW PTR TO MS BYTE OF DIVIDEND (& EVENTUAL RESULT)
  000A 7912         158             MOV     R1,#12H         ;R1 NOW PTR TO R2 THIS BANK SO CAN USE NIBBLE XCHD INSTRUCTION
  000C E6           159             MOV     A,@R0           ;GET MS BYTE OF DIVIDEND
  000D 8BF0         160             MOV     B,R3            ;PUT NIBBLE DIVISIOR IN B
  000F 84           161             DIV     AB
  0010 85F012       162             MOV     12H,B           ;MOVE REMAINDER TO R2 SO CAN USE NIBBLE EXCHANGE IN LOOP
  0013 F6           163             MOV     @R0,A           ;MS BYTE NOW DONE, STORE IN DIVIDEND/RESULT
  0014 08           164             INC     R0              ;BUMP PTR TO NEXT MS BYTE OF DIVIDEND
  0015 7F02         165             MOV     R7,#2           ;LOOP COUNTER FOR REMAINING BYTES OF DIVIDEND
                    166     DN1:
  0017 E6           167             MOV     A,@R0           ;GET NEXT BYTE OF DIVIDEND IN ACC (WE ONLY NEED MS NIBBLE)
  0018 D7           168             XCHD    A,@R1           ;PUT REMAINDER STORED AT R2 IN LO NIBBLE OF ACC
  0019 C4           169             SWAP    A               ;NOW REMAINDER IS IN HI NIBBLE OF ACC & NEXT MS NIBBLE OF
                                                            ;  DIVIDEND IS IN LO NIBBLE
  001A 8BF0         170             MOV     B,R3            ;DIVIDE AGAIN BY DIVISOR NIBBLE
  001C 84           171             DIV     AB
  001D 85F012       172             MOV     12H,B           ;AGAIN SAVE REMAINDER TO R2 THIS BANK FOR LO NIBBLE EXCHANGE
  0020 C4           173             SWAP    A               ;PUT NIBBLE RESULT IN HI NIBBLE AND STORE IN DIVIDEND/RESULT
  0021 C6           174             XCH     A,@R0           ;  WHILE RETRIEVING ORIGINAL DIVIDEND BYTE (WE ONLY NEED LS
                                                            ;  NIBBLE)
  0022 C4           175             SWAP    A               ;LS NIBBLE OF DIVIDEND BYTE TO HI NIBBLE OF ACC FOR XCHD
  0023 D7           176             XCHD    A,@R1           ;REMAINDER TO LO NIBBLE OF ACC
  0024 C4           177             SWAP    A               ;SWAP REMAINDER TO HI NIBBLE FOR NEXT DIVIDE
  0025 8BF0         178             MOV     B,R3            ;DIVIDE AGAIN BY DIVISOR NIBBLE
  0027 84           179             DIV     AB
  0028 85F012       180             MOV     12H,B           ;AGAIN REMAINDER TO R2 THIS BANK FOR NIBBLE SWAP
  002B D6           181             XCHD    A,@R0           ;STORE LO NIBBLE RESULT THIS BYTE IN DIVIDEND/RESULT
  002C 08           182             INC     R0              ;BUMP PTR FOR NEXT DIVIDEND BYTE
  002D DFE8         183             DJNZ    R7,DN1          ;FINAL TWO BYTES DONE?
                    184
  002F 85F0E0       185             MOV     ACC,B           ;PUT NIBBLE REMAINDER FROM LAST DIVIDE IN ACC FOR CALLER
  0032 D0D0         186             POP     PSW             ;RESTORE REGISTERS AND SWITCH BACK TO CALLERS REGISTER BANK
  0034 D0F0         187             POP     B
  0036 22           188             RET
                    189
                    190     END
```